Optimal Sample Patches Selection for Tile-Based Texture Synthesis

Weiming Dong
ISA-ALICE, INRIA Lorraine
Weiming.Dong@loria.fr

Shuangxian Sun
Beijing Jiaotong University
Shuangxian.Sun@hotmail.com

Jean-Claude Paul
CNRS/Tsinghua University
paul@tsinghua.edu.cn

Abstract

One significant problem in tile-based texture synthesis is the selection of sample patches. The reason is that the tiling process may produce results with poor quality when the sample patches used to fill the tiles can not find good cutting pathes at the merging stage. In this paper, we use Genetic Algorithm to find the best group of sample patches from a considerable number of patch groups. Using tile set gene rated by these optimal sample patches can significantly improve the result quality of the tile-based texture synthesis.

Keywords: Genetic Algorithm, Tile-Based Texture Synthesis, Wang Tiles, Image Quilting

1. Introduction

Texture synthesis is one of the hottest topics in the field of Computer Graphics, Computer Vision and Image Processing. It is important for rendering synthetic images and animations. The goal of texture synthesis can be stated as follows [19]: Given a texture sample, synthesize a new texture that, when perceived by a human observer, appears to be generated by the same underlying stochastic process.

Nowadays neighborhood-based methods are popularly used in texture synthesis, including point-based techniques [19, 4, 1] and patch-based techniques [3, 9, 20, 12, 8]. Hybrid texture synthesis method has also already been developed [13]. These methods attempt to synthesize larger textures by copying selected regions (pixels or patches) of the sample texture and, in some way, obscuring region boundaries. Most of them can produce high quality results, but the speeds are still slow. They can't avoid the heavy searching process in order to get a suited pixel or patch for the output image.

Recently, several real time texture synthesis algorithms were presented. Liang et al. [10] use quadtree pyramid, optimized kd-tree and principal components analysis (PCA) to accelerate the patch placement process, but simply alpha-blends the overlap regions (feathers). Zelinka and Garland [21] present a two-phases method towards real-time texture synthesis. They first examine the input texture at length, building a data structure which is called a *jump map* to store several similar pixels for every pixel in the input texture, then only a random pixel choice according to the probability in the jump map is needed in the synthesis phase. Wang et al. [16] extend the *jump map* technique to patch-based maps and get better results than the previous one. Cohen et al. [2] present a new stochastic algorithm to non-periodically tile the plane with a small set of *Wang Tiles*. Using this tiling technique they can synthesize as much non-periodic texture as needed at run time. It is also a real-time texture synthesis algorithm, but the quality of the results is not good enough due to the simple sample patches selection method in the tiles filling process. Wei [18] develop a new tiling technique with GPU to improve the tile-based texture mapping method, but the result quality also can not be ensured.

In this paper, we present a near global optimal sample patches selection method based on *Genetic Algorithm* (GA) to improve the tile set generation process of [2]. With it and the tiling method of [2] we form an improved real-time texture synthesis algorithm and can produce better results than the original method.

2. Related Work

We briefly explain the idea of *Wang Tiles*-based texture synthesis technique and the *Genetic Algorithm* (GA).

Wang Tiles (or *Wang Dominoes*), first proposed by Hao Wang in 1961 [17], are equal-sized squares with a color on each edge which give rise to a simple undecidable decision problem. A valid tiling requires all shared edges between tiles to have matching colors. When a set of *Wang Tiles* are filled with texture patterns that are continuous across matching tile edges, a valid tilling from such a set can produce an arbitrarily large texture without pattern discontinuity.

To create a continuous texture from a sample, textures must be found for each tile that fit together across the boundaries with matching colors. In [2], a tile is created by combining diamond shaped (squares rotated by 45 degrees) sample portions of the source image, one for each

edge color of horizontal and vertical edges. The set of *Wang Tiles* are generated automatically. First, the required number of square sample images (for example 6 for 18 tiles) are selected from the source image, one sample for one color. Then construct each tile by combining the four sample diamonds that correspond to the edge colors of it. In this case, they find four cutting paths to combine the four samples to form the tile using the optimization algorithm given in [3]. Next, the four combined samples that together have a diamond shape are cut along their diagonals. This forms the final tile which is slightly smaller than the sample diamonds due to the edge color fitting requirement [2]. At the run time synthesis stage, all the work need to do is only to copy the tile texture to the corresponding place in the output tiling.

Genetic Algorithm (GA) [14, 7] is an efficient stochastic search method for solving optimization problem. It is so named as the scheme is based on the mechanics of natural selection and natural genetics. Research interests in heuristic search algorithms with underpinnings in natural and physical processes began in the 1970s, when Holland [6] proposed *Genetic Algorithm* (GA). GA generates a sequence of populations using a selection mechanism, and applies crossover and mutation as search mechanisms. GA has demonstrated considerable success in providing good solutions to many complex optimization problems [11], such as capital budgeting, vehicle routing problem, critical path problem, parallel machine scheduling, redundancy optimization, open inventory network etc. The advantage of GA is due to its ability to obtain a global optimal solution fairly in a multidimensional search landscape, which has several locally optimal solutions as well. During the search process, it can automatically achieve and accumulate the knowledge about the search space, and adaptively control the search process to approach the global optimal solution.

3. Discussion on Wang Tiles-Based Texture Synthesis Algorithm

The artifacts of the *Wang Tiles*-based texture synthesis method are clear. First, as addressed in [2], because too few sample patches of the original texture are used, the random perception of the final image is not good. This problem can be solved by simply increasing the number of tiles. Our work still use this method to overcome the first artifact. All of our results are generated with 18 tiles. Second, due to the artifacts introduced by the Efros's quilting algorithm, the quality of the final image is not very good even when compared with the results of the original algorithm in [3]. The main reason of this problem is because the tile construction algorithm can not ensure that every four samples used to construct one tile is optimized selected, as in [3]. When the overall quilting error of the four samples is high, a tile with poor quality is generated, and it will directly affect the quality of the final image. In [2], this problem is partly resolved by iterating over sets of samples. During the iteration, for each set, calculate the sum of the pixel color errors that occur along all cutting paths, until a set with small overall error is found or the iterative time reaches a preset number. This technique can definitely increase the quality of the final results, but new problems are introduced: First, the performance of this iteration method is poor, because it is difficult to balance the requirements of both result optimization and searching efficiency. It is not easy to set a fitted error threshold to reach a good set within a small number of iterations. Second, this method can not ensure that the samples in the result set are global optimized. The set with a tile of very high quilting error and another of comparatively small error can also meet the threshold of the overall error. Apparently this kind of sets can not produce good result.

Our GA-enhanced tile set generation algorithm can effectively resolve these problems.

4. GA-Enhanced Tile Set Generation Algorithm

GA starts with an initial set of random-generated chromosomes called a population where each chromosome is a coding of a solution of an optimization problem. All chromosomes are evaluated by an evaluation function which is some measure of fitness. A selection process based on the fitness values will form a new population. The cycle from one population to the next is called a generation. In each new generation, all chromosomes will be updated by the crossover and mutation operations. Then the selection process selects chromosomes to form a new population. After performing a given number of cycles, or other termination criteria is satisfied, we denote the best chromosome into a solution, which is regarded as the optimal solution of the optimization problem. In our algorithm, we use GA to find the near global optimal source image sample patches to generate the tile set.

4.1. Improved Patch Matching Algorithm

Image quilting [3] can do an excellent job on many textures, but has difficulty with some structured textures. Usually boundary artifacts are visible, as shown in Figure 1. In order to overcome this problem, we develop a new patch matching algorithm for *image quilting*, which is similar to the method in [15], but not exactly the same.

We note that many of the boundary artifacts are caused by the reason that the neighboring patches do not have a smooth boundary transition between them. As addressed in [15], this would occur if the overlap sum of squared differences (SSD) are small, but there is no good minimum

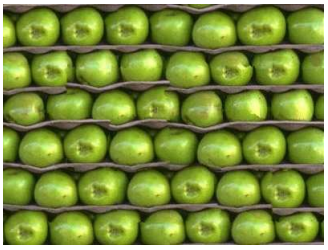

Figure 1. Boundary artifacts in a structured texture

cutting path to use as the boundary. We extend the matching algorithm by considering not only the SSD, but also the cost of the minimum cutting path. After determining the SSD (denoted by $D_{overlap}$) between the current overlap region and an input patch being considered, calculate the luminance (the *Y* parameter of the YIQ color model [5]) errors of pixels along the minimum cutting path (denoted by D_{path}). Divide the SSD by the number of pixels (denoted by $n_{overlap}$) in the overlap region, and divide the path error by the number of pixels (denoted by n_{path}) along the path. Then the distance used for comparing this patch with the other potential matches is:

$$D_{boundary} = \omega \cdot \frac{D_{overlap}}{n_{overlap}} + 3 \cdot (1-\omega) \cdot \frac{D_{path}}{n_{path}}$$

where $\omega \in (0,1)$ is a weighted factor to calculate the final distance value. We only use *Y* parameter here because human's eyes are much more sensitive to luminance than the chromaticity information (hue and purity) [5]. This can accelerate the calculation of the minimum cutting path errors.

Figure 2 shows a comparison of our algorithm with *image quilting* method [3]. One can see that with our patch matching algorithm the boundary artifacts in the output images are effectively reduced.

4.2. GA-Enhanced Sample Patches Selection

The original problem of the tile set generation is to select *n* sample patches from the source image and use them to fill the *m* tiles (for example $n = 4$ when $m = 8$, $n = 6$ when $m = 18$). Here we use GA to find the optimal *n* sample patches:

Initialization: To ensure an optimal solution be obtained in a reasonable runtime, an initial population consists of a considerable amount of chromosomes (*n* sample patches) is necessary. To start the algorithm, an integer *pop_size* is defined as the number of chromosomes. From the source image, *pop_size* chromosomes are randomly selected, denoted by $A_1, ..., A_{pop\_size}$. Every chromosome contains *n* sample patches (*n* genes) selected from the source image:

$$A_i = (g_i^1, g_i^2, ...g_i^n), i = 1, 2, ..., pop\_size$$

Evaluation: In GA, the selection of chromosomes to reproduce is determined by a probability assigned to each chromosome A_i. This probability is proportional to its fitness relative to other chromosomes in the population, i.e. chromosomes with higher fitness will have more chance to produce offsprings by the selection process. In the context of sample patches selection, the fitness of a chromosome, is evaluated by an evaluation function, $E(A_i)$, which measures the performance of the sample patches derived from that chromosome. This evaluation function, in essence, computes the overall boundary error and is defined as:

$$D_{sum}^i = \sum_{1 \le j \le 4} D_{boundary}^j, i = 1, 2, ..., m$$

$$V_{sum} = Variance(D_{sum}^1, D_{sum}^2, ..., D_{sum}^m)$$

$$E(A_i) = \omega \cdot \sum_{1 \le i \le m} D_{sum}^i + (1-\omega) \cdot V_{sum}$$

where D_{sum}^i is the overall boundary error of one tile in the tile set which is generated by the sample patches derived from the chromosome A_i. V_{sum} is the variance of all the tile overall boundary errors, we add this factor in order to protect the global quality of the final output image. So the final evaluation function is the weighted sum of the overall boundary errors of the tile set and its variance, the boundary errors of each tile are all generated using the algorithm in Section 4.1, $\omega \in (0,1)$ is the weighted factor.

To obtain an optimal tile set, we need to determine the best sample patches that has the least overall boundary error. Hence, the fitness function $F(A_i)$ for selection is defined as the inverse of the evaluation function, i.e. $F(A_i) = \frac{1}{E(A_i)}$. based on the value of the fitness function for each chromosome, the population of chromosomes $A_1, A_2, ..., A_{pop\_size}$ can be rearranged from high fitness to low fitness.

Selection: The selection process is basically a "spinning the roulette wheel" process. The roulette wheel is spun *pop_size* times and each time a chromosome from the rearranged population $A_1, A_2, ..., A_{pop\_size}$ is selected. As we have stated, the chromosome with higher fitness should have a higher probability to be selected. This is achieved by the following steps:

Step 1: Define a ranking function for each chromosome

$$eval(A_i) = \alpha(1-\alpha)^{(i-1)}, i = 1, 2, ..., pop\_size$$

where $\alpha \in (0,1)$ is a predefined parameter.

Figure 2. Improved image quilting results. Left to right: input texture sample, image quilting results, our results.

Step 2: Based on this ranking function, calculate the cumulative probability, q_i for each chromosome A_i is given by

$$q_0 = 0; q_i = \sum_{j=1}^{i} eval(A_j), i = 1, 2, ...pop\_size$$

Step 3: Generate a random real number γ in $(0, q_{pop\_size}]$.

Step 4: Select the i^{th} chromosome A_i such that $q_{i-1} < \gamma \leq q_i$. Repeat Step 3 and Step 4 until *pop_size* copies of chromosomes are obtained. These *pop_size* copies of selected chromosomes $\hat{A}_1, \hat{A}_2, ..., \hat{A}_{pop\_size}$ are the mother chromosomes for the reproduction of the next generation.

Crossover: The crossover process will produce a new generation of population based on the set of mother chromosomes $\hat{A}_1, \hat{A}_2, ..., \hat{A}_{pop\_size}$ resulting from the selection process. We define $P_c \in [0, 1]$ as the probability of crossover. Hence, the expected number of mother chromosomes that will undergo crossover is $P_c \cdot pop\_size$. To pick the parents for crossover, we perform the following action:

For $i = 1$ to *pop_size*
  generate a random number γ;
  if $\gamma < P_c$ put $\hat{A}_i$ in a parent list
  else put $\hat{A}_i$ in a non-parent list
end.

We denote the parent list as $\tilde{A}_1, \tilde{A}_2, ...$ and the non-parent list as $\bar{A}_1, \bar{A}_2, ...$ If there are odd number of members in the parent list, the last member will be switch to the non-parent list. With an even number of member in the parent list, we group the members into pairs $(\tilde{A}_1, \tilde{A}_2), (\tilde{A}_3, \tilde{A}_4), ...$ A random natural number $c \in [2, n]$ is generated and applied to the crossover operation to each parent pair to produce two children given by:

$$X = (\tilde{g}_1^1, ..., \tilde{g}_1^{c-1}, \tilde{g}_2^c, ..., \tilde{g}_2^n)$$
$$Y = (\tilde{g}_2^1, ..., \tilde{g}_2^{c-1}, \tilde{g}_1^c, ..., \tilde{g}_1^n)$$

A new generation of population is produced by combining the children produced by the parent pairs and the non-parent chromosomes.

Mutation: In GA, to avoid the solution being bounded by a local optimum, a mutation process is applied to the chromosomes in the new generation. We define $P_m \in [0, 1]$ as the probability of mutation. Hence, the expected number of chromosomes that will undergo mutation is $P_m \cdot pop\_size$. Similar to the picking of chromosomes for crossover, chromosomes are picked for mutation based on P_m. For each chromosome picked, denoted by $A_l = (g_l^1, g_l^2, ...g_l^n)$, two mutation position n_1 and n_2 is randomly chosen where $1 \leq n_1 < n_2 \leq n - 1$. For $j = n_1$ to n_2, change g_l^j to a random patch from the source image which does not equal to any of the existing genes. After all the genes in and between the two mutation positions are changed, A_l changes to form a new mutated chromosome A_l'.

Termination: Two termination criteria are used. Either the process is executed to produce a fixed number of generations and the best solution among all these generations is chosen, or the process is terminated if no further improvement in the best solution is observed in eight consecutive generations.

5. Results and Conclusions

We present an optimal tile set generation algorithm by applying the *Genetic Algorithm* to the sample patches selection stage of the *Wang Tiles*-based texture synthesis algorithm. And to get a good cutting path for the tile filling operation, we also improve the patch matching algorithm of *image quilting*. This can effectively reduce the boundary artifacts of the results. The whole working flow of our enhanced tile-based texture synthesis algorithm is: First randomly initialize a considerable number of sample

patch groups (as the chromosomes) from the source image, then use *Genetic Algorithm* to find the best one, with our new patch matching criteria. Finally use the same method in [2] to fill the tile set and generate the final images (tiling). Our method can be nicely applied in the environment where real-time texture synthesis is needed, while *image quilting* can not (need seconds or minutes to generate an image).

Figure 3 shows a comparison of our algorithm with the original *Wang Tiles*-based method [2] and the image quilting method [3]. We can see that the quality of our results is comparable with the *image quilting* algorithm and is better than the original *Wang Tiles*-based algorithm. Figure 4 shows some more synthesis results of our algorithm, all the results are 256×256, the tile number of the tile set is 18. For *Genetic Algorithm*, the following parameter values are used: $pop\_size = 50$, $\alpha = 0.05$, $P_c = 0.3$, $P_m = 0.2$, and the number of generation is 200. The GA training time for the images in Figure 4 is shown in Table 1 (Pentium IV 3.2GHz PC, 1GB RAM). We can see that our GA training algorithm is very efficient for finding the optimal sample patches.

Real-time texture synthesis is still an open research topic now. In the future, we want to find some more effective methods to increase the quality of the final results without affecting the real-time performance.

References

[1] M. Ashikhmin. Synthesizing natural textures. In *SI3D '01: Proceedings of the 2001 symposium on Interactive 3D graphics*, pages 217–226, New York, NY, USA, 2001. ACM Press.

[2] M. F. Cohen, J. Shade, S. Hiller, and O. Deussen. Wang tiles for image and texture generation. *ACM Trans. Graph.*, 22(3):287–294, 2003.

[3] A. A. Efros and W. T. Freeman. Image quilting for texture synthesis and transfer. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 341–346, New York, NY, USA, 2001. ACM Press.

[4] A. A. Efros and T. K. Leung. Texture synthesis by non-parametric sampling. In *ICCV '99: Proceedings of the International Conference on Computer Vision-Volume 2*, page 1033, Washington, DC, USA, 1999. IEEE Computer Society.

[5] D. Hearn and M. P. Baker. *Computer graphics: C version*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, second edition, 1997.

[6] J. H. Holland. *Adaptation in natural and artificial systems*. University of Michigan Press, Ann Arbor, Michigan, USA, 1975.

[7] J. R. Koza. Survey of genetic algorithms and genetic programming. In *Proceedings of 1995 WESCON Conference*, pages 589–594. IEEE, 1995.

[8] V. Kwatra, I. Essa, A. Bobick, and N. Kwatra. Texture optimization for example-based synthesis. *ACM Transactions on Graphics, SIGGRAPH 2005*, August 2005.

[9] V. Kwatra, A. Schödl, I. Essa, G. Turk, and A. Bobick. Graphcut textures: image and video synthesis using graph cuts. *ACM Trans. Graph.*, 22(3):277–286, 2003.

[10] L. Liang, C. Liu, Y.-Q. Xu, B. Guo, and H.-Y. Shum. Real-time texture synthesis by patch-based sampling. *ACM Trans. Graph.*, 20(3):127–150, 2001.

[11] B. Liu and B. Liu. *Theory and Practice of Uncertain Programming*. Physica-Verlag, 2002.

[12] Y. Liu, W.-C. Lin, and J. Hays. Near-regular texture analysis and manipulation. *ACM Trans. Graph.*, 23(3):368–376, 2004.

[13] A. Nealen and M. Alexa. Hybrid texture synthesis. In *EGRW '03: Proceedings of the 14th Eurographics workshop on Rendering*, pages 97–105, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[14] M. Srinivas and L. M. Patnaik. Genetic algorithms: A survey. *Computer*, 27(6):17–26, 1994.

[15] N. Vavra. Texture quality extensions to image quilting. CS766 Final Projec of University of Wisconsin, 2002.

[16] B. Wang, J. Yong, and J. Sun. Real-time texture synthesis with patch jump maps. In *International Symposium on Computational and Information Sciences(CIS'04)*, pages 1155–1160, 2004.

[17] H. Wang. Proving theorems by pattern recognition ii. *Bell System Technical Journal*, 40:1–42, 1961.

[18] L.-Y. Wei. Tile-based texture mapping on graphics hardware. In *HWWS '04: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 55–63, New York, NY, USA, 2004. ACM Press.

[19] L.-Y. Wei and M. Levoy. Fast texture synthesis using tree-structured vector quantization. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 479–488, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[20] Q. Wu and Y. Yu. Feature matching and deformation for texture synthesis. *ACM Trans. Graph.*, 23(3):364–367, 2004.

[21] S. Zelinka and M. Garland. Towards real-time texture synthesis with the jump map. In *EGRW '02: Proceedings of the 13th Eurographics workshop on Rendering*, pages 99–104, Aire-la-Ville, Switzerland, Switzerland, 2002. Eurographics Association.

Table 1. Genetic Algorithm training time

Texture Name	Tile Number	Patch Number	Patch Size	Overlap Size	GA Training Time (Seconds)
Leaves	18	6	80	16	6
Fabric	18	6	120	20	12
Mud	18	6	120	20	11

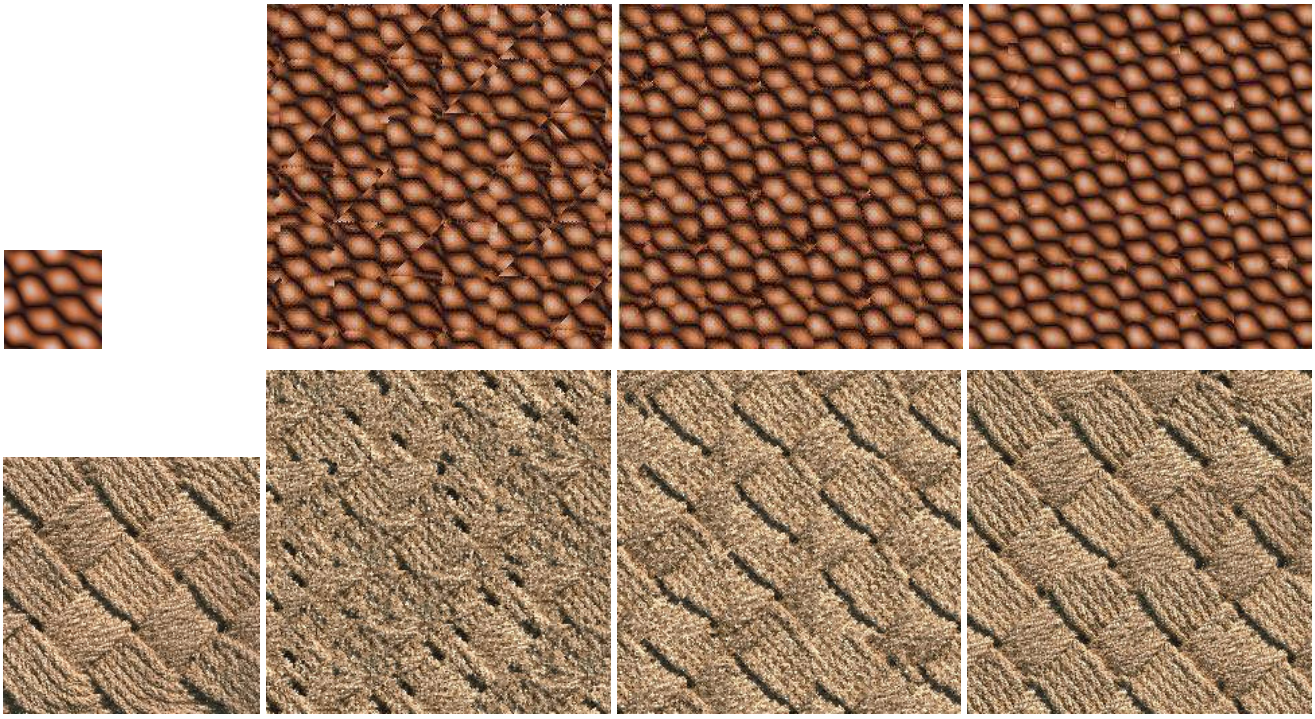

Figure 3. Comparison among Wang Tiles-based method (left), our method (middle) and image quilting method

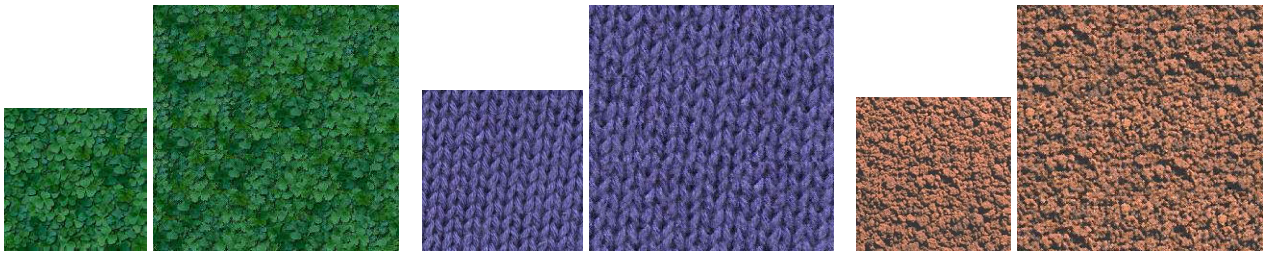

Figure 4. More results of our synthesis algorithm. In each example, the small (left) image is the input sample texture and the large (right) image is the output texture.